Claims

1. A method for modifying software,
- wherein initially, on the vendor side, from an original piece of software consisting only of source text (SC1, SC2, SC), a hybrid form of said original software is formed in such a way that at least one part (SC1) of the source text is compiled into a byte or binary code (B1,..,B) and at least one further part (SC2) of the source text is converted into a code (CodeML) formulated in a meta markup language for at least one variation point (VP2,..VP),
- wherein subsequently, on the customer side, only at least one variation point (VP2) of the hybrid form (SW) of the original software is converted as necessary by means of a transformation (T) in accordance with transformation rules (TR) into at least one other code (CodeML*) formulated in the meta markup language and
- wherein said other code (CodeML*) directly forms a modified variation point (VP2*) of an adapted piece of software (SW*) or a source code (SC*) is formed from said other code (CodeML*) by means of a converter (RCONV) and then a binary or byte code (B*) of the modified variation point (VPB2) of an adapted piece of software (SW*) is formed by means of a compiler (COMP), with the original (SW) and the adapted software (SW*) differing in terms of their program execution and/or program content.

2. The method as claimed in claim 1,
wherein the transformation rules (TR) have at least one modification rule for a variation point.

3. The method as claimed in claim 1 or 2,
wherein the modification rule initiates an update to a more recent software version or a patching operation.

4. The method as claimed in claim 1 or 2,

wherein the modification of at least one variation point (VP) is performed by means of the transformation at runtime.

5. The method as claimed in one of the preceding claims, wherein the programming language of the source code is Java and the meta markup language of the variation points is XML and wherein the transformation and the rule description are implemented by means of XSLT and XSL.

6. A method for modifying source code,
- wherein a first code (CodeML) formulated in a meta markup language with language extensions (LE) formulated in at least one meta markup language is available as the source code,
- wherein the source code is converted by means of a transformation (T) in accordance with transformation rules (TR) into a second code (CodeML*) formulated in the meta markup language without the language extensions (LE) formulated in the meta markup language,
- wherein the transformation rules form a language converter (LC) which resolves or applies the language extensions (LE) of the first code in such a way that they can be processed by a back-converter (RCONV) that has no corresponding language extension, and
- wherein said second code can be converted into a second source code (SC*) formulated in the first programming language or a different programming language and yields a valid binary code/byte code (B*).

7. The method as claimed in claim 6,
wherein at least one language extension (LE*) is newly generated in the second code (CodeML*) or taken over from the first code (CodeML), and this generation or takeover is performed by the language converter (LC).

8. A method for modifying source code,
- wherein a source code (SC) formulated in a first programming language is converted into a first code (CodeML) formulated in a meta markup language,

- wherein a transformation (T) takes place exclusively in accordance with transformation rules TR into a second code (CodeML*) formulated in the meta markup language and
- wherein said second code is transformed into a second source code (SC*) formulated in the first programming language or a different programming language, the first and the second source code differing in terms of their functionality (B, B*).

9. The method as claimed in claim 8,
wherein the transformation rules (TR) include at least one condition C and one logic component L and/or code fragment CF itself.

10. The method as claimed in claim 8 or 9,
wherein transformation rules (TR) include at least one fragment in the form of a template (TPF) and/or at least one pattern (PF) in which at least one code modification is effected with the aid of the transformation T.

11. The method as claimed in claim 10,
wherein TR is embodied in such a way that a mechanism for backing up at least one system state is incorporated into the second source code (CodeML*) with the aid of the transformation T in order to enable a migration into other versions.

12. The method as claimed in claim 8 or 9,
wherein at least one template (TP) is formed from the first code or a fragment of the first code with the aid of the transformation T.

13. A method for modifying source code,
- wherein a source code (SC) formulated in a first programming language is converted into a first code (CodeML) formulated in a meta markup language,
- wherein an item of information (INFO) formulated in the meta markup language and influencing the subsequent program execution (B*) is added by means of a transformation (T) to

said first code in a substituting or non-substituting way and in this way the second code (CodeML*) also formulated in the meta markup language is formed, the transformation being performed in accordance with transformation rules (TR) formulated in a transformation description language, and
- wherein said second code is transformed into a second source code (SC*) formulated in the first programming language or a different programming language, the program content and/or program execution (B) of the first source code (SC) differing from the program content and/or program execution (B*) of the second source code (SC*).

14. The method as claimed in claim 13,
wherein said information (INFO) includes at least one code fragment (CFb) and
wherein the second source code is formed in that at least one code fragment (CFa) contained in the first source code is replaced with the aid of the transformation by the at least one code fragment (CFb) contained in the fragment.

15. The method as claimed in claim 13,
wherein the information (INFO) specifically includes data (D) in the form of initialization states (SSDb) or state data (SDa) or database data (Dc).

16. The method as claimed in claim 15,
wherein the transformation rules (TR) are influenced by said data (D).

17. The method as claimed in one of the claims 13 to 16,
wherein data (D) and/or code fragments (CF) are additionally embedded in the transformation rules.

18. The method as claimed in one of the claims 13 to 17,
wherein data generated by checkpoints is added by means of a transformation in such a way that the internal state of the original program is available at the restart of the program or can be used by said program.

19. The method as claimed in one of the claims 13 to 17, wherein information (INFO) includes updates or patches.

20. The method as claimed in claim 13 to 17, wherein fragments (LF1, LF2) contain internationalization information which serves for the adaptation to different natural languages.

21. The method as claimed in one of the claims 13 to 17, wherein data and/or code fragments originate from a library and
carry information tailored to customers or customer groups.

22. An arrangement for modifying software,
- wherein a hybrid form of an original piece of software is present in such a way that at least one part (SC1) of a source text is compiled into a byte or binary code (B1,..,B) and at least one further part (SC2) of the source text is converted into a code (CodeML) formulated in a meta markup language for at least one variation point (VP2,..VP),
- wherein a device for transformation (T) is present in such a way that only at least one variation point (VP2) of the hybrid form (SW) of the original software can be converted as necessary by means of the transformation (T) in accordance with transformation rules (TR) into at least one other code (CodeML*) formulated in the meta markup language,
whereby said other code (CodeML*) directly forms a modified variation point (VP2*) of an adapted piece of software (SW*) or a source code (SC*) can be formed from the other code (CodeML*) by means of a converter (RCONV) and then a binary or byte code (B*) of the modified variation point (VPB2) of an adapted piece of software (SW*) can be formed by means of a compiler (COMP) and whereby the original (SW) and the adapted software (SW*) differ in terms of their program execution and/or program content.

23. An arrangement for modifying source code,

- wherein a processor is present in such a way that a transformation (T) of the source code (CodeML) is performed in accordance with transformation rules (TR) formulated in an extended style description language and a language converter (LC) contained therein in such a way that either a second code (CodeML*) formulated in the meta markup language without the language extensions (LE) of the first code (CodeML) that were formulated in the meta markup language,
or a second code (CodeML*) formulated in the meta markup language is generated using new and/or the original language extensions (LE) formulated in the meta markup language, and
- wherein a back-converter (RCONV) is present in such a way that said second code is transformed into a source code (SC*) formulated in the first programming language or a different programming language.

24. An arrangement for modifying source code,
- wherein a first converter (CONV) is present in such a way that a source code (SC) formulated in a first programming language is converted into a first code (CodeML) formulated in a meta markup language,
- wherein a processor is present in such a way that the CodeML is converted by means of a transformation (T) exclusively in accordance with transformation rules (TR) into a second code (CodeML*) formulated in the meta markup language and
- wherein a second converter (RCONV) is present in such a way that said second code is converted into a second source code (CodeML*) formulated in the first programming language or a different programming language, the first and the second source code differing in terms of their functionality and/or content (B, B*).

25. An arrangement for modifying source code,
- wherein a first converter (CONV) is present in such a way that a source code (SC) formulated in a first programming language is converted into a first code (CodeML) formulated in a meta markup language,

- wherein a processor is present in such a way that an item of information (INFO) formulated in the meta markup language and influencing the subsequent program execution (B*) is added by means of a transformation (T) to said first code in a substituting or non-substituting manner and in this way the second code (CodeML*) also formulated in the meta markup language is formed, the transformation being performed in accordance with transformation rules (TR) formulated in a transformation description language, and
- wherein a second converter (RCONV) is present in such a way that said second code is transformed into a second source code (SC*) formulated in the first programming language or a different programming language, the program content and/or program execution (B) of the first source code (SC) differing from the program content and/or program execution (B*) of the second source code (SC*).